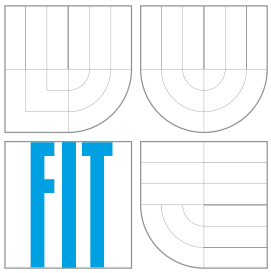
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTELIGENTNÍ RSS ČTEČKA

INTELLIGENT RSS READER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ŽOUŽELKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. SVATOPLUK ŠPERKA
SUPERVISOR

BRNO 2010

Abstrakt

Práce je zaměřená především na problematiku týkající se technologií pro publikaci a odběr webového obsahu jako jsou například RSS nebo Atom. Hlavním cílem bylo vytvoření aplikace využívající těchto služeb, tedy programu s možnostmi pro stažení, zpracování a zobrazení uživateli požadované zprávy. Výsledkem je plnohodnotná RSS čtečka, implementovaná v jazyce Java, poskytující mnohé zajímavé funkce.

Abstract

This work is especially aimed at technologies for publication and extracting web content like for example RSS and Atom. The main goal was creation of an application that makes use of these services in order to download, treat and presentate required messages to user. The result of this work is full-valued RSS reader implemented in Java language which provides many interesting functions.

Klíčová slova

RSS, Atom, syndikace obsahu webu, XML

Keywords

RSS, Atom, web content syndication, XML

Citace

Martin Žouželka: Inteligentní RSS čtečka, bakalářská práce, Brno, FIT VUT v Brně, 2010

Inteligentní RSS čtečka

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Svatopluka Šperky. Uvedl jsem všechny literární prameny a díla, ze kterých při tvorbě tohoto dokumentu vycházel.

.....
Martin Žouželka
19. května 2010

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Svatopluku Šperkovi za jeho cenné rady, připomínky a odbornou pomoc v souvislosti s řešením této práce.

Obsah

Kapitola 1

Úvod

Internet patří v dnešní době k nejpoužívanějším médiím poskytujícím nejrůznější informace svým uživatelům. Ty jsou většinou dostupné v podobě webových stránek. Vzhledem k velkému množství těchto dat se začínají používat i jiné techniky pro publikování a odběr konkrétních novinek pomocí fomátů určených k syndikaci obsahu. Důsledkem toho existuje snaha o sledování a agregaci těchto zpráv nesoucích velký informační potenciál.

Cílem práce je vytvořit klientskou desktopovou aplikaci, která by byla schopna fungovat jako plnohodnotný agregátor novinek s pokročilými funkcemi pro vyhledávání, stahování a zobrazení zvolených RSS zpráv z internetu. Důraz je kladen také na návrh uživatelského prostředí a realizaci funkcí s tím spojených. Jednotlivé ovládací prvky musí pracovat tak, aby užívání aplikace bylo co možná nejintuitivnější. Je třeba vytvořit určitý systém, podle kterého se dají konkrétní novinky snadno dohledat a který nabízí možnost kategorizace zpráv. V neposlední řadě musí program navenek působit profesionálním, avšak moderním dojmem. Výsledná aplikace by měla být alespoň částečně konkurenceschopná s jinými programy stejného druhu.

1.1 Struktura práce

Nejprve se zaměříme na zařazení používaných formátů pro syndikaci obsahu a také se stručně podíváme na jejich historii, kde je zmíněno několik podstatných událostí, které výrazným způsobem ovlivnili vývoj až po jejich současnou podobu.

Třetí kapitola je již zaměřena na analýzu problému samotného, tedy zjištění, jaké informace lze pomocí zmíněných formátů získat a jak se s nimi dá naložit. Je zde nastíněna struktura nejpoužívanějších syndikačních zdrojů společně s vysvětlením jednotlivých značek.

Následující čtvrtá kapitola pak uvádí konkrétní programový návrh jak po funkční stránce (specifikace funkcí a možností aplikace), tak po prezentační (rozložení komponent a ovládacích prvků GUI). Dále je zde popsán výsledný objektový model a způsoby archivace článků.

V páté kapitole se již dostáváme k samotné implementaci aplikace v jazyce Java, kde je rozebrána většina problémů, které se v souvislosti s ní vysktytly. Poslední podkapitola pak pojednává o testování programu a jejich výsledcích.

Na závěr je provedeno zhodnocení práce jako celku s výhledem na další možná rozšíření. V příloze lze nalézt kompletní programovou dokumentaci ve formě *JavaDoc*u a také zdrojové kódy aplikace společně s výslednou sestavenou a spustitelnou verzí aplikace.

Kapitola 2

Historie syndikačních formátů

Než se dostaneme k analýze problému samotného, řekneme si něco o formátech a technologiích, ze kterých budeme později v tomto dokumentu vycházet. Následujících několik podkapitol nám vysvětlí význam jednotlivých zkratek a také se zaměří na zařazení formátů do příslušných jazykových rodin. Dozvíme se něco o historii syndikace obsahu od jejích počatků až po současná řešení a možnosti jejího využití.

2.1 Zařazení formátů

V úvodu se objevila zmínka o RSS zprávách. Před vysvětlením tohoto pojmu se nejprve zmíníme o množinách jazyků, do kterých syndikační formáty spadají, abychom si je mohli dát do souvislostí s jinými známými fakty.

2.1.1 Rodina značkovacích jazyků

Značkovací jazyky hrají v dnešní době velmi výraznou roli na poli informačních technologií. Díky svým vlastnostem jsou používány k mnoha účelům. Jedná se o speciální typ jazyka určeného zejména pro popis dokumentů. Převládá v nich nestrukturovaný text, do kterého jsou vloženy strukturované úseky pomocí značek. Pro každou značku musí vždy existovat speciální posloupnost znaků. Značky se zpravidla dělí na *závorkové*, které jsou tvořeny počáteční a uzavírající značkou a *sekvenční*, které jsou pouze jediné.

Dále lze značkování dělit na:

- **Procedurální**, při kterém mají všechny značky pevný význam (například HTML).
- **Univerzální**, neboli popisné značkování, kde nemají značky pevně danou sémantiku. Ta je dána až v průběhu zpracování značek (například XML)[10].

2.1.2 Jazyk XML

Mezi značkovací jazyky patří i jakyk XML (*eXtensible Markup Language*), který ve většině případů slouží jako platforma pro přenos nejrůznějších dat a metainformací. Tento jazyk není majetkem žádného komerčního subjektu. Původně byl vytvořen konsorciem W3C podle zkušeností s přechozími značkovacími jazyky.

Jedná se v podstatě o velmi vhodně navržený dokument pro ukládání strukturovaného a semi-strukturovaného textu určeného pro šíření a publikaci na celé řadě médií. Jde o otevřený formát, který neklade jeho uživatelům žádné podmínky nebo hranice. Můžeme si

nadefinovat vlastní značky dle libosti. Aplikace zpracovávající tyto značky pak využívají definovaných jmenných prostorů, které určují, jaké značky jsou používány. Přestože kořeny XML spadají do oblasti publikování, tento jazyk je vhodný i pro identifikaci komplexních datových struktur, které nebudou nikdy prohlíženy ani tištěny.

Dokument XML obsahuje logickou a fyzickou strukturu. Logická struktura dělí dokument do jednotek a podjednotek, nazývaných elementy. Fyzická struktura pak dovoluje uložit a pojmenovat samostatně části dokumentu, zvané entity, v různých datových souborech, aby mohly být tyto informace opětovně použity a aby bylo možné vkládat odkazy na data neodpovídající standardu XML. Například každá část knihy může být reprezentována elementem, který obsahuje další elementy popisující tabulku či obrázek, ale obrazová data jsou uložena v samostatných souborech entit.

Logická struktura obsahuje řadu omezení, která musí zůstat zachována, aby se dal považovat dokument za validní. K zamezenení zmatku v pojmenování elementů existují prostředky, jako například DTD nebo XML schéma, které definují povolené prvky. Parsery těchto dokumentů pak porovnávají strukturu konkrétního XML s DTD, zda odpovídá požadavkům [7].

2.2 Co se skrývá pod zkratkou RSS?

Konečně se dostáváme k vysvětlení samotného pojmu RSS. V podstatě jde o XML dokument speciálně navržený pro uveřejňování a také odebírání internetových zpráv a novinek. Tyto zprávy mohou být různorodého charakteru, avšak logická struktura dokumentu má určitá pravidla v závislosti na použité verzi. Syndikační formát RSS obsahuje úplný nebo zestručněný obsah článku a také metadata, jako například autorství, datum vydání a jiné. Zkratka RSS se vždy odvíjí od konkrétní verze a bude vysvětlena v následující podkapitole.

2.3 Historie RSS a Atom

Informační technologie patří k nejdynamičtěji se rozvíjejícím odvětvím, a tudíž nelze očekávat, že zmíněné služby mají dalekosáhlou minulost. RSS se dá považovat za poměrně zaběhlou techniku publikování a odebíraní obsahu. Naproti tomu formát Atom je novinkou v tomto směru, avšak v mnohém svého staršího rivala předčí.

2.3.1 Počátky RSS formátu

Vznik RSS je datován od roku 1997, kdy byl vytvořen první koncept pro *Resource Description Framework* (RDF). Značkovací jazyk RDF byl používán k ukládání metadat, tedy informací o informacích (autor článku, vydavatel, atd.). V roce 1999 firma Netscape vytvořila formát s označením RSS 0.90 (*RDF Site Summary*), který byl postaven na prvních návrzích RDF. O další rozšíření se zasloužil její zaměstnanec Dan Libby a následně vyšla upravená verze 0.91. Avšak ve stejné době Dave Winer z firmy Userland také upravil původní formát a pojmenoval ho stejným způsobem, tedy RSS 0.91 (*Rich Site Summary*). Tato situace byla pro mnohé vývojáře poněkud zavádějící, protože existovaly dva stejně pojmenované formáty s různou specifikací. Firma Netscape nakonec opustila vývoj těchto technologií a přenechala volnou ruku ostatním.

Zajímavým milníkem se stal vznik verze 1.0, kterou vyvinula společnost O'Reily. Nová specifikace byla založena na RDF standardu a velmi se lišila od původních verzí 0.91 a 0.92.

V důsledku zaměření na RDF se stala nová verze nekompatibilní s původními, což vedlo k velkému zmatku na trhu.

Firma Userland vzhledem k těmto okolnostem vyvinula svůj vlastní formát, který byl kompatibilní s původními verzemi 0.91 a 0.92 a pojmenovala jej RSS 2.0 (*Really Simple Syndication*). V roce 2003 Dave Winer předal vlastnictví RSS 2.0 pod správu Berkmanova centra pro Internet a společnost spadající pod právnickou fakultu Harvardské university a tím „zmrazil“ další úpravy specifikace tohoto formátu.[4].

2.3.2 Počátky Atom formátu

I přes značné pokroky v oblasti vývoje RSS nejnovější verze 2.0 stále postrádá některé důležité vlastnosti. Do obsahové části RSS nelze umístit HTML značky v normálním tvaru, je nutné je escapovat. Jako další nedostatek se jeví to, že MIME typ RSS `application/rss+xml` není registrován v IANA databázi a nepodporuje pokročilé XML kryptografické mechanismy [15].

Vzhledem k těmto a jiným nedostatkům, v červnu roku 2003 Sam Ruby založil diskusní stránku, kde lidé spojení s touto oblastí definovali požadavky na vznik nového formátu, který by nahradil RSS. Projekt ze začátku dostal různá označení a jména, avšak nakonec se ustálil název Atom.

První verze se objevila jako Atom 0.2 v červenci roku 2003. V prosinci téhož roku vyšla verze 0.3, která získala silnou podporu od firmy Google, která jej zařadila hned do několika svých známých webových služeb jako Blogger nebo Gmail. V následujícím roce se začalo diskutovat nad standardizací formátu, tedy jeho zařazením mezi standardy W3C (*World Wide Web Consortium*) nebo IETF (*Internet Engineering Task Force*). V červnu 2004 byla vytvořena skupina zvaná Atompub (*Atom Publishing Format and Protocol Working Group*), která dala formátu ucelené tělo a ten vyšel pod označením Atom 1.0. V srpnu téhož roku byl Atom zařazen do IETF [3].

2.4 Využití syndikace obsahu dnes

V dnešní době nabízí službu syndikace obsahu velké množství webových stránek a portálů. K nim se přidaly i nejrůznější internetové diskusní skupiny a blogy. Lidé pracující nebo zajímající se o oblast webových aplikací a služeb dobře znají oba zmíněné formáty a využívají jejich možností, avšak zatím se nedá říci, že by šlo o masovou záležitost. I přesto lze na internetu nalézt velké množství klientských aplikací určených k odebírání a agregaci novinek, které nabízejí svým uživatelům velmi zajímavé funkce. Kromě desktopových programů existuje také celá řada webových čteček. Agregátor zpráv lze mít přímo v internetovém prohlížeči v podobě doplňku, který je možno jednoduše nainstalovat. Možností a forem využití syndikace obsahu dnes existuje opravdu velmi mnoho.

Kapitola 3

Analýza problému

Po stručném pojednání o historii vývoje zmíněných formátů je nutné poznat, jak jsou tyto dokumenty strukturovány a jaké informace nám vlastně poskytují. Z těchto poznatků budeme později vycházet při navrhování a implementaci programu.

3.1 Struktura RSS kanálu

Nejprve je třeba si uvědomit, že pro RSS formát existuje více verzí a od naší aplikace se očekává, že bude schopna zpracovat i ty zastaralejší a méně používané. Vzhledem k těmto okolnostem je nutností znát rozdíly ve specifikacích těchto verzí. Naštěstí se jejich tvůrci až na verzi 1.0 snažili o zpětnou kompatibilitu s předchozími typy a tím zjednodušili práci lidem vytvářejícím parsery těchto kanálů.

Tato podkapitola neobsahuje úplně všechny verze RSS, ale pouze ty, které se dneska nejčastěji používají. Podle průzkumu verze 0.91 je používaná v 10% případů, 1.0 v 11% a 2.0 v 76% [9].

3.1.1 Verze 0.91

Nejjednodušší, avšak s řadou omezení. K těm největším patří to, že žádný zdroj nemůže obsahovat více jak 15 článků. Jak již bylo zmíněno, z tohoto základu vycházela i nejnovější verze 2.0.

- **Element <rss>** - hlavní prvek rss zdroje. Jeho součástí jsou všechny ostatní značky a jejich obsah. V atributu `version` udává svou verzi.
  - **Element <channel>** - obsahuje všechny informace potřebné k nastavení RSS zdroje. Každý rss element musí obsahovat právě jednu značku channel.
    - **Element <title>** obsahuje název článku.
    - **Element <link>** je povinný u prvků `<channel>`, `<image>`, `<item>`, `<textinput>`. Obsahuje odkaz na jeden z těchto zdrojů.
    - **Element <description>** obsahuje popis zdroje, který většinou pojednává o poskytovateli této služby.
    - **Element <language>** určuje, ve kterém jazyce jsou články napsány (platí pro celý dokument). Použitý jazyk je zde reprezentován svým unikátním kódem.

* **Element <copyright>** je volitelným prvkem verze 0.91. Obsahuje informace o autorských právech vztahujícím se ke zdroji.
* **Element <pubDate>** obsahuje datum zveřejnění kanálu.
* **Element <lastBuildDate>** udává datum poslední aktualizace kanálu. Podle této informace lze zjistit, jak často autor svůj veřejný zdroj obměňuje.
* **Element <docs>** ve svém obsahu zahrnuje dodatečné informace o kanálu, jako URL adresu stránky nebo popis kanálu.
* **Element <image>** udává informace o obrázku zdroje. Musí obsahovat další prvky `<title>`, `<url>`, `<link>`. Mezi volitelné značky patří `<description>`, `<widht>` a `<height>`.
* **Element <managingEditor>** většinou obsahuje e-mailovou adresu správce **obsahu** zdroje. Kromě adresy mohou být použity i jiné kontaktní informace.
* **Element <webMaster>** uchovává kontakty spojené s osobou zodpovědnou za **technickou** správu zdroje.
* **Element <rating>** nebývá příliš využíván, avšak v některých případech se může v dokumentu vyskytnout. Poskytuje informace ohledně hodnocení RSS zdroje pomocí standardu PICS (Platform for Internet Content Selection).
* **Element <skipHours>** je dalším nepříliš užívaným prvkem. Pomocí něj můžeme stanovit v jakém časovém rozmezí nebude zdroj aktualizován. Obsahuje další značky s názvem `<hour>`.
* **Element <skipDays>** má v podstatě stejný význam jako předchozí prvek. Obsahuje další prvky `day`.
* **Element <item>** reprezentuje jeden článek v rámci zdroje. Ve verzi 0.91 je možno použít pouze 15 těchto prvků. V pozdějších verzích bylo tohle omezení zrušeno. Stejně jako element `<channel>` má několik povinných a nepovinných elementů.
  - **Element <title>** obsahuje název článku. Jelikož první verze RSS neobsahovaly metainformace jako ID článku, čtečky si musely vytvořit vlastní unikátní klíč ke každému článku a právě pomocí názvu článku bylo třeba zjistit, jestli tento již nebyl dříve jednou stažen a zpracován.
  - **Element <link>** je povinným prvkem elementu `<item>`. Obsahuje URL adresu odkazující se na celý obsah článku. Pro RSS čtečky se jedná o důležitou informaci, pomocí které můžou čtenáři poskytnout kompletní obsah zprávy.
  - **Element <description>** v sobě nese text se stručným výtahem z článku, avšak mnohdy obsahuje i celý obsah a HTML formátování. Mnohe čtečky jej renderují přímo jako HTML stránku a zobrazují i příslušné odkazy a obrázky. Neobsahuje žádné další potomky nebo atributy.
  - **Element <textinput>** nabízí možnost odebíratelům kanálu pokládat autorovi dotazy a vést s ním diskuzi. Tento element se momentálně příliš nepoužívá, protože se dává přednost přehledným komentářům přímo na stránce se zprávou. Později byl prvek nahrazen značkou `<comments>`, která odkazuje na část stránky zprostředkovávající tuto diskuzi. Obsahuje další elementy jako `title`, `description`, `name`, `link`.

3.1.2 Verze 1.0

Tato verze je úzce spjata s jazykem RDF a značně se liší od 0.91 a 2.0. Lze tvrdit, že se jedná o slepou větev vývoje RSS, nicméně některé webové zdroje ji stále používají. Organizace a uspořádání jednotlivých značek jsou jiné, než tomu bylo u předcházející verze. Výsledkem byla zpětná nekompabilita a kritika z řad odborníků. Tvůrci zmíněného zdroje viděli budoucnost hlavně v jazyce RDF, na jehož základě byla verze 1.0 postavena.

Hlavním kořenovým elementem je zde značka `<rdf:RDF>`, ze které je patrné, že používá jmenný prostor RDF. Jedním z jejich potomků je element `<channel>`, který již neobsahuje všechny zbylé značky, ale pouze metainformace spojené s nastavením používání kanálu. Na druhou stranu je zde novinka v podobě seznamu s přehledem článků v prkvu `<items>`. Po uzavírací značce `</channel>` následuje výčet jednotlivých článků tak, jak je již známe z předchozí verze.

Jednotlivé články stále postrádají jeden důležitý údaj, a to datum zveřejnění. Základ verze 1.0 nenabízí možnost tento údaj připojit, avšak existuje rozšíření v podobě jmenného prostoru *Dublin Core*, který má řadu vestavěných elementů. Tedy pokud definujeme jmenný prostor `dc` na začátku dokumentu, můžeme využít jeho prvku `<dc:date>`. Pomocí této informace je možné řadit a filtrovat jednotlivé články podle času jejich publikace, což je pro aplikace využívající RSS syndikace podstatné.

3.1.3 Verze 2.0

V současnosti se jedná o suverénně nejpoužívanější formát pro syndikaci obsahu, a to i přestože Atom je v mnoha ohledech navržen lépe. Pokud odebíráte novinky z nějakého zdroje, s velkou pravděpodobností je to právě verze 2.0. Vznikla jako následník verze 0.92, která je potomkem zmíněné 0.91. Ta nabízí hned několik novinek. Jako reakce na poptávku po zdroji, který by dokázal odkazovat na nejrůznější mediální přílohy, se zde poprvé objevila značka `<enclosure>`, která umožňuje syndikaci podcastingu.

Tvůrci tohoto formátu se snažili vzít ty nejlepší vlastnosti z 0.91 a 0.92 a doplnit známé nedostatky. Počet článků již není nijak limitován. Odkazy již nejsou omezeny pouze na protokoly `http` a `ftp`, ale lze použít prefixy jako `https://`, `news://` či `mailto://`. Dále bylo přidáno několik nových elementů. V rámci prvku `<channel>` jsou to:

- **Element <category>** poskytuje možnost autorovi RSS kanálu přiřadit jej do různých kategorií. Předpokládalo se, že čtečky budou tyto metainformace hojně využívat, avšak vzhledem k velkému množství kategorií, které poskytují autoři, bývá pro uživatele složité se v nich orientovat, a tak se spíše přihlíží k možnosti, aby odebíratel zařadil daný kanál do kategorie podle svého uvážení. Prvek může obsahovat i potomka `<domain>` obsahující adresu stránky s použitými kategoriemi.

- **Element <generator>** je další volitelný prvek v rámci elementu `<channel>`. Obsahuje identifikační popis programu, s jehož pomocí byl kanál vytvořen. Existuje velké množství editorů specializovaných na vytváření RSS kanálů, které zde automaticky vkládají své informace.

- **Element <cloud>**, pod jehož názvem se nemyslí nic spojeného s mrakem, avšak podpora rozhraní *rssCloud*. RSS kanály jsou aktualizovány nedříve jednou za hodinu, nicméně to někdy nemusí stačit. Nabízí se zde ještě možnost využít webovou aplikaci typu *cloud*, která nám zajistí rychlejší aktualizaci. Více informací o této službě lze nalézt na [1].

- **Element <ttl>** znamená, stejně tak jako u síťových packetů, *Time To Live*. Ovšem v tomto případě údaj určuje počet minut, po který by měl kanál být uložen ve vyrovnávací paměti, než bude znovu aktualizován.

Nyní se zaměříme na nové prvky spadající do elementu `<item>`.

- **Element <author>** byl vytvořen jako reakce na stále větší poptávku po kontaktu autora článku. Nejčastěji obsahuje jeho e-mailovou adresu.
- **Element <category>** má tu samou funkci jako stejnojmenný prvek v rámci celého kanálu. Poskytuje možnost autorovi zařadit článek do různých specifických kategorií, podle kterých je možno novinky třídit. Může obsahovat libovolný počet kategorií, což bývá v některých případech až kontraproduktivní. Opět může mít jednoho potomka s názvem `<domain>` odkazujícím na stránku s popisem jednotlivých kategorií.
- **Element <enclosure>** patří mezi velmi zajímavé prvky. Pomocí něj lze využít tzv. podcastingu, tedy možnosti přiřadit ke zdroji nejrůznější soubory multimediálního charakteru. Mezi ně patří jak hudební, tak soubory obsahující videozáznamy, obrázky, aplikace, modely a jiné. Tato technologie se velmi rychle uchytila hlavně na stránkách internetových rádií, které prostřednictvým RSS informují posluchače o nových nahrávkách a možnostech jejich zakoupení nebo stáhnutí. Musí povinně obsahovat 3 následující atributy:
  - **Atribut url** odkazující na zdroj MIME souboru na internetu. Existuje zde omezení pro odkazy, a to na používání http protokolu.
  - **Atribut length** určuje velikost souboru v bytech.
  - **Atribut type** udávající MIME typ souboru. Většinou je zde uveden typ multimediálního formátu a kodek nebo metoda komprese. Například pro hudební soubor s příponou *.mp3* je to `audio/mpeg`.
- **Element <guid>** je pro RSS čtečky velmi důležitým údajem, který funguje jako primární klíč ke každému článku. Pokud je v kanálu obsažen, čtečka podle něj může jednoduše zjistit, zda již byl daný článek dříve zpracován a nemusí prohledávat ostatní články podle názvu. Je nutné, aby tento řetězec byl v rámci všech článků zcela unikátní. Z tohoto důvodu se většinou udává URL adresa stránky s obsahem článku, která je jedinečná. Jiné zdroje tvoří obsah tohoto elementu odlišným způsobem. Například pomocí náhodně vygenerovaných čísel nebo znaků. Existují návody, jak zajistit takovému řetězci unikátnost, avšak ve většině případů se používá zmíněná URL. Prvek může obsahovat jeden atribut `<isPermaLink>`, který, pokud obsahuje hodnotu true, značí, že je použita právě URL.
- **Element <source>** je užitečný zejména, když daný zdroj obsahuje články z více jiných zdrojů. Pomocí tohoto prvku můžeme určit z jakého originální zdroje článek pochází. Může obsahovat atribut `url` s adresou zdroje.

V RSS 2.0 lze samozřejmě definovat své vlastní jmenné prostory a používat své značky. Stačí pouze definovat nějaký jmenný prostor, do kterého budou patřit. V mnoha případech RSS zdroje vkládají celý, nebo částečný obsah zprávy do značky `<content:encoded>`, který pak může obsahovat HTML prvky ve svém normálním tvaru, což je užitečné, protože prvek `<description>` smí obsahovat pouze escapované HTML značky.

3.2 Struktura Atom kanálu

Nyní se zaměříme na strukturu nováčka v oblasti syndikace obsahu zvaného Atom. Jak již bylo na začátku zmíněno, první verze Atomu vyšla až po několika odnožích RSS, takže jeho autoři vycházeli z předešlých zkušeností s RSS formátem.

Atom má hned několik výhod oproti RSS, a to především jasné specifikace, o jaký druh obsahu se jedná. Může to být například HTML, XHTML, čistý text, nebo i obsah v binárním tvaru. Další změnou je, že všechny prvky atomu patří do jmenného prostoru `Atom1.0`. Existuje i zde možnost definice vlastních jmenných prostorů. V Atomu lze použít i relativní URI adresy za pomocí atributu `xml:base`, což u RSS zkrátka nešlo. Kanál typu RSS není možno nijak validovat, protože neexistuje žádné DTD nebo XML schéma, podle kterého by se dalo ověření správnosti provést. V případě Atomu existuje pro tyto účely ISO standard *RelaxNG* schéma. Případné další porovnání těchto dvou formátů lze nalézt zde [14].

- **Element <feed>** je hlavním prvkem obsahujícím všechny ostatní elementy.
  - **Element <id>** funguje jako unikátní klíč, s jehož pomocí jsme schopni jednotlivé kanály rozlišovat. Tento prvek je v rámci Atom specifikace povinný.
  - **Element <title>** obsahuje název kanálu. Většinou se uvádí název portálu nebo společnosti, která vlastní autorská práva ke zdroji.
  - **Element <updated>** uchovává časovou známku, kdy byl kanál aktualizován.
  - **Element <subtitle>** udává dodatečný popis ke kanálu. Většinou se jedná o stručné pojednání o tom, k jakému tématu se obsažené články vážou.
  - **Element <link>** zahrnuje URI adresu, ze které byl kanál získán. Většinou obsahuje ještě další atributy jako `type` určující typ obsahu, `hreflang` s jazykem, ve kterém jsou informace ke kanálu napsány a `title` s názvem.
  - **Element <author>** patří mezi prvky obsahující metadata ohledně autora kanálu. Může obsahovat další značky jako `<name>`, `<email>` a `<uri>`.
  - **Element <contributor>** udává další případné spolupracovníky podílející se na tvorbě Atom kanálu. Stejně jako u prvku `<author>` i zde se nachází elementy `<name>`, `email`, `<uri>`.
  - **Element <entry>** reprezentuje jeden článek v rámci kanálu. Může obsahovat další dodatečné informace o článku nebo o jeho autorech. Opět zde můžeme rozdělit jednotlivé prvky na povinně se vyskytující a nepovinné. Do první této skupiny patří, stejně tak jako u prvku `<feed>`, značky `<id>`, `<title>` a `<updated>`. Všechny tyto informace a mnohé další lze dohledat na stránkách se specifikací Atomu [13].
    - **Element <id>** udává identifikační klíč. Kanál může obsahovat i více článků se stejným názvem, avšak s unikátním klíčem. Nejčastěji se zde vyskytuje permanentní URI adresa obsahu článku.
    - **Element <title>** obsahuje název článku.
    - **Element <updated>** udává datum a čas, kdy byl článek aktualizován.
    - **Element <author>** určuje autora článku. Tak, jako jeho jmenovec pro prvek `<feed>`, může obsahovat další značky `<name>`, `<email>` a `<uri>`.

* **Element <summary>** v sobě skrývá stručný výtah z článku, nebo přímo kompletní článek. Může vlastnit atribut `type`, který specifikuje, o jaký typ obsahu se jedná.
* **Element <category>** poskytuje autorovi článku možnost zařadit jej do určité kategorie. Většinou se zde objevují značky jako `term`, `scheme` a jiné.
* **Element <generator>** plní v podstatě stejnou funkci jako u RSS kanálu. Lze jej využít zejména pro odstraňování chyb při testování kanálu.
* **Element <icon>** tvoří IRI odkaz, který identifikuje obrázek nebo ikonu patřící k článku. Ten pak čtečky mohou uživateli zobrazit pro snadnější orientaci mezi články.
* **Element <logo>** má v podstatě stejný význam jako předcházející prvek.
* **Element <published>** obsahuje datum, kdy byl článek poprvé zvěřejněn.
* **Element <rights>**, jak již jeho název napovídá, nás informuje o autorských právech, která se na článek vztahují.
* **Element <source>** plní tutéž roli, jako stejnojmenný prvek v RSS. Pokud jsou články zdroje sestaveny z více jiných kanálů, tento element nás odkáže na ten původní.
* **Element <content>** je jedním z nejdůležitějších prvků zprávy, který v sobě uchovává obsah článku. Jak již bylo zmíněno na začátku této podkapitoly, může zahrnovat více typů obsahu. Od HTML formátování až po binární podobu. Má několik užitečných atributů jako například `type`, který právě určuje druh použitého obsahu. Dalším z nich je atribut `src`, který odkazuje na adresu, ze které obsah pochází.

Kapitola 4

Návrh aplikace

Nyní, když už známe strukturu syndikačních formátů, následuje kapitola zaměřená na návrh samotné aplikace. Nejprve byla vytvořena specifikace požadavků na výsledný program a od této se začaly odvíjet jednotlivé fáze vývoje. Bylo třeba také vymyslet, jak budou získaná data z XML uchovávána v paměti a jakým způsobem se k nim bude přistupovat.

4.1 Návrh uživatelského rozhraní

Tato podkapitola se zaobírá jednou z nejdůležitějších problematik projektu, a to uživatelským prostředím.

V názvu této práce se vyskytuje slovo inteligentní, což napovídá, že výsledný program, kromě své vlastní aplikační logiky, bude muset být schopen inteligentě komunikovat s jeho uživatelem. Mezi základní požadavky patří zejména intuitivní ovládání, rychlá a nenáročná interakce s uživatelem a v neposlední řadě také příjemný vzhled působící na člověka moderně a pozitivně. Doba konzolových aplikací pro širší veřejnost je již nenávratně pryč, a proto je třeba orientovat se především na nové technologie v oblasti GUI (*Graphical User Interface*). Naštěstí v dnešní době existuje celá řada vývojových prostředí, které obsahují nástroje pro zjednodušení tvorby uživatelských rozhraní a není nutné pro vývojáře začínat od nuly. To ovšem neznamená, že tyto nástroje za nás odvedou všechnu práci. Nyní se zaměříme na funkce a možnosti, které by měla ve výsledku aplikace uživateli poskytovat.

4.1.1 Uživatelské kategorie

Téměř od každé RSS čtečky se očekává, že svému uživateli poskytne možnost zařadit si zdroje do svých kategorií a vytvořit si tak svou vlastní hierarchii kanálů. Bylo by nejspíše zklamáním pro uživatele, kdyby tato možnost v aplikaci chyběla. Některé RSS zdroje poskytují informace o kategoriích, do kterých je zařadil jejich autor, avšak většina z nich působí spíše zavádějícím způsobem. Pomocí kategorií, které si uživatel sám vytvoří, je možno provádět příslušné operace nejen nad jednotlivými kanály, ale přímo nad jejich skupinami. Složka nebo kategorie zde v tomto případě figuruje jako skupina zdrojů, která nabízí stejné možnosti jako samostatný kanál. Kupříkladu, pokud by čtenář chtěl aktualizovat všechny kanály dané kategorie, stačí tuto operaci provést právě nad ní.

Samozřejmostí je také pojmenování kanálu podle svého uvážení. Původní názvy bývají často zbytečně dlouhé a nejednoznačné. Avšak při prvním stažení by čtečka měla nabídnout původní pojmenování, aby nenutila uživatele okamžitě vymýšlet svůj název.

V grafickém prostředí bývá tohle všechno realizováno pomocí komponenty zobrazující určitou stromovou hierarchii. Jednotlivé uzly lze rozkliknout a zobrazit tak jejich obsah. Také by bylo vhodné zde realizovat pokročilejší techniky přesouvání mezi složkami, tedy přetahování kurzorem myši (*Drag & Drop*).

4.1.2 Základní operace s kanály a články

Uživatel musí mít možnost přídávat, odebírat a určitým způsobem modifikovat jeho strom kanálů. K tomu samozřejmě patří i možnost aktualizovat vybrané zdroje a zjistit tak, zda neobsahují nové články. Každý objekt reprezentující jeden kanál pak musí v sobě uchovávat adresu s odkazem na zdroj, ze kterého se aktualizuje, aby uživatel tuto nemusel opětovně zadávat. Vzhledem k velkému množství metadat v každém souboru s kanálem můžeme poskytnou uživateli i další informace o daném kanálu. Některé čtenáře může zajímat popis daného RSS zdroje nebo autorská práva vztahující se na něj. Všechny tyto akce musí být přístupny jak z hlavního menu umístěného pod vrchním okrajem okna aplikace, tak z kontextového menu vyvolaného stisknutím pravého tlačítka myši nad označeným kanálem. Pro vyšší intuitivnost se v uživatelském prostředí bude nacházet ještě panel s tlačítky, jejichž ikona bude odpovídat příslušné operaci. V konečné podobě tedy mezi tyto akce patří:

- Přidat kanál.
- Stáhnout kanál.
- Upravit nastavení kanálu.
- Zobrazit informace o kanálu.
- Odebrat kanál.

Co se týče článků, většina čteček nabízí možnost jejich dalšího kategorizování nebo označování. Vzhledem k tomu, že některé zdroje poskytují opravdu velké množství zpráv, je tato funkce pro čtenáře užitečná. Uživatel pak bude moci v rámci jednotlivých skupin články vyhledávat a filtrovat.

První takovouto kategorií by měla být skupina článků, které si uživatel ještě nepřečetl. Tedy články, které uživatel stáhl z internetu, avšak je zatím neshlédl. Jenomže jak zjistit, zda odebíratel danou zprávu opravdu četl? Dá se pouze předpokládat, že když si uživatel nechá zprávu zobrazit, tak že ji přečetl. To ovšem nemusí být vždy pravda, a tak by zde měla existovat možnost opětovně označit článek jako nepřečtený.

Řazení článků pouze do skupin mezi přečtené a nepřečtené nemusí zcela vždy vyhovovat. Uživatel může chtít nějakým způsobem danou zprávu odlišit od ostatních tak, aby ji vždy rychle nalezl. Z tohoto důvodu byla zavedena kategorie „důležité“. Kromě odlišení se od ostatních zpráv má ještě jednu zajímavou funkci spojenou s archivací článků, která bude popsána později.

Dále zde musí existovat možnost zobrazení celého obsahu článku. Většina RSS čteček nabízí spuštění příslušného odkazu v externím prohlížeči. Některé pokročilejší aplikace dokáží stránku zobrazit přímo v sobě, avšak jejich engine pro renderování HTML kódu ne vždy zvládne zobrazit náročnější obsah. U operací s články by tedy nemělo chybět:

- Zobrazit obsah článku v aplikaci (za použití specializovaného enginu).
- Zobrazit obsah článku v externím prohlížeči.

- Smazat článek
- Označit článek jako důležitý.
- Označit článek jako nepřečtený.

4.1.3 Vyhledávání

Určitě znáte ten nepříjemný pocit, když hledáte určitou věc a nemůžete si vzpomenout, kde naposledy byla. Tento případ se stává často i při práci na počítači s určitou aplikací. Pokud chceme uživateli ušetřit čas a námahu, musíme mu v GUI poskytnout spolehlivé vyhledávací nástroje tak, aby to, co hledá (nejspíše určitý článek nebo část jeho obsahu), našel co možná nejrychleji.

Poněvadž již byly zavedeny kategorie zdrojů a skupiny článků, můžeme poměrně efektivně jednotlivé zprávy filtrovat a zúžit tak okruh hledání. I přesto by měl mít uživatel ještě navíc možnost v tomto okruhu vyhledávat určité slovo nebo část řetězce, který se vyskytuje v názvu nebo obsahu článku. Spojení těchto metod vytváří již poměrně účinný nástroj k vyhledávání i ve velkém množství zpráv.

V uživatelském prostředí se dá tohle realizovat pomocí dvou jednoduchých komponent. Nejprve potřebujeme textové pole pro zadání vyhledávaného výrazu a potom roletové menu pro výběr skupiny, do které zpráva patří. V seznamu s články se pak budou průběžně objevovat pouze ty, které vyhovují zadání. Aby byla interakce co nejrychlejší, tabulka s články by se měla automaticky překreslovat při každé změně výrazu v textovém poli.

4.1.4 Rozvržení ovládacích prvků

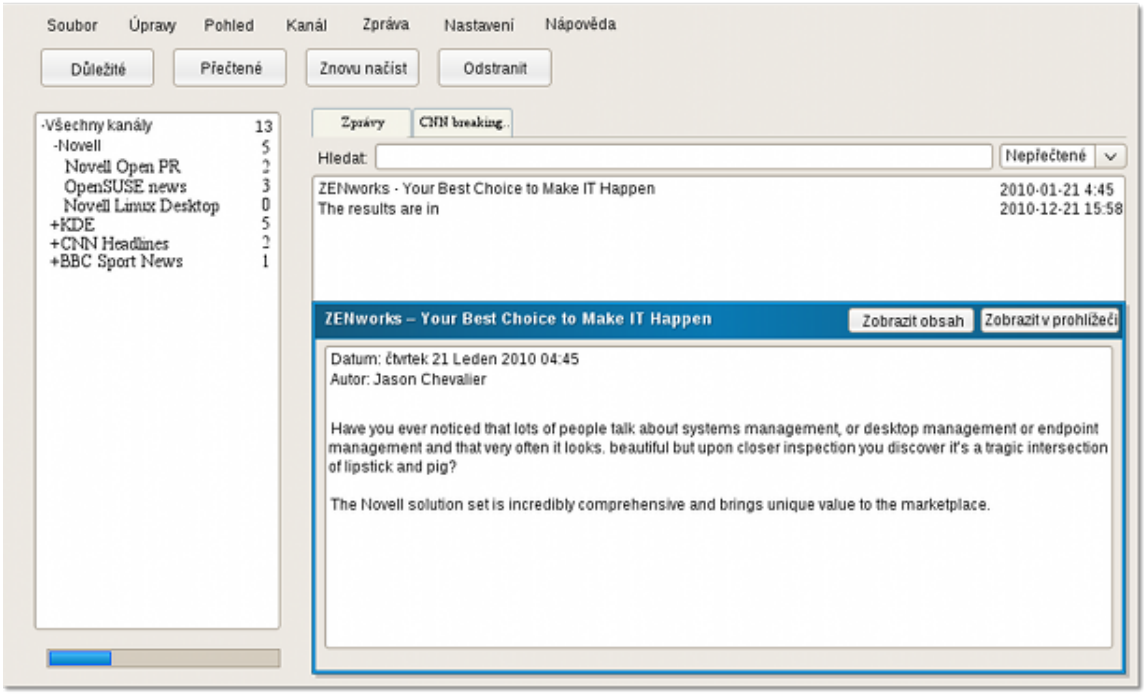
Výsledná aplikace bude obsahovat velké množství komponent, které lze rozdělit do 3 základních sekcí. První z nich je zaměřená na práci se zdroji, kterou bude představovat hierarchický strom s kategoriemi a kanály. Ta bývá zpravidla umístěná v levé části hlavního okna. Další sekcí je seznam či tabulka s články, která bude zobrazovat zprávy přidružené k označenému kanálu nebo skupině kanálů a vyhovující nastaveným filtrům. Podle konvencí bývá v pravé horní části hlavního menu. Poslední sekce bude mít za úkol zobrazovat obsah označeného článku. Vzhledem k tomu, že RSS a Atom soubory mohou obsahovat HTML formátování, jako nejlepší řešení se zde nabízí použití komponenty, která tento obsah dokáže správně vykreslit. Tato část GUI bývá většinou ukotvena k pravému spodnímu okraji okna. Každý z těchto tří prostorů si může uživatel zvětšovat nebo zmenšovat na úkor těch ostatních pomocí posuvníku.

Existují i jiné možnosti uspořádání těchto prvků. Některé RSS čtečky jako GreatNews[1] nebo Sage[2] zobrazují seznam článků jako vygenerovanou HTML stránku, na které se orientujeme výhradně pomocí hypertextových odkazů. Tento způsob je zajímavý, avšak při vetším množství článků se může stát seznam zpráv poměrně nepřehledným. Vygenerovaná stránka je pak příliš dlouhá a při vyhledávání článků se zobrazují výsledky sekvenčně. Další způsob rozřazení komponent nabízí například čtečka NetNewsWire[3], kde je možno mít všechny 3 sekce vedle sebe ve svislých sloupcích. Tento způsob rozložení je také zajímavý, avšak pro obsahovou část zde zbývá poměrně úzký pruh místa. To ovšem nemusí znamenat problém pro majitele moderních širokoúhlých monitorů.

[1] http://www.curiostudio.com/
[2] http://sagerss.com/
[3] http://netnewswireapp.com/

Ještě ovšem schází vyřešit jeden problém spojený s umístěním komponenty, která bude schopna zobrazovat celkový obsah webové stránky. Po vzoru ostatních čteček se jako nejlepší řešení nabízí vytvoření nové záložky, která bude poskytovat dostatek místa k vykreslení stránky s článkem.

Po všech těchto úvahách je návrh rozložení komponent hotov (viz obr. 4.1).



Obrázek 4.1: Skica návrhu uživatelského prostředí

4.2 Objektový model

Programy s uživatelským rozhraním bývají v dnešní době nejčastěji vyvíjeny za pomocí objektově orientovaného přístupu. Celá aplikace se pak skládá z objektů, které mezi sebou komunikují pomocí implementovaných metod. Tyto principy jsou podporovány a hojně využívány mnoha moderními programovacími jazyky.

Nejprve je třeba vytvořit určitý nákres nebo alespoň představu o tom, jaké typy objektů bude aplikace potřebovat a jak mezi sebou budou komunikovat. Tento proces se nazývá tvorba objektového modelu programu. Je důležité tuto část nepodceňovat, protože jakákoliv pozdější změna v návrhu znamená předělání základů, na kterých je aplikace postavena. To pak většinou vede k přepisování velkého množství již napsaného kódu. Specifikace jednotlivých používaných typů kanálů již byla rozebrána na začátku této kapitoly. Nyní je nutné pečlivě uvážit, které prvky z této množiny využijeme a do jaké objektové třídy budou

náležet.

Celé logické uspořádání tříd bylo provedeno do dvou hlavních vrstev. Později, po výběru implementačního jazyka, byly tyto vrstvy reprezentovány Java balíky. První vrstva se stará o prezentaci a interakci s uživatelem. Druhá vrstva je tvořena třídami obsahujícími atributy a metody pro práci s daty. Obrázek 4.2 ukazuje výsledný diagram tříd a vazby mezi nimi.

4.2.1 Prezentační vrstva

Tato skupina tříd představuje celkové uživatelské rozhraní aplikace se všemi dialogy pro komunikaci s uživatelem. K tomu samozřejmě náleží i třídy starající se o datové modely a rendery složitějších komponent (třídy určené k vykreslení komponent). V obrázku 4.2 nejsou kvůli přehlednosti tyto třídy zahrnuty.

Výchozím prvkem je třída MainWindow, která bude tvořit hlavní okno aplikace. Ostatní modální a nemodální dialogy jí budou předávat získané a validované informace ze vstupu.

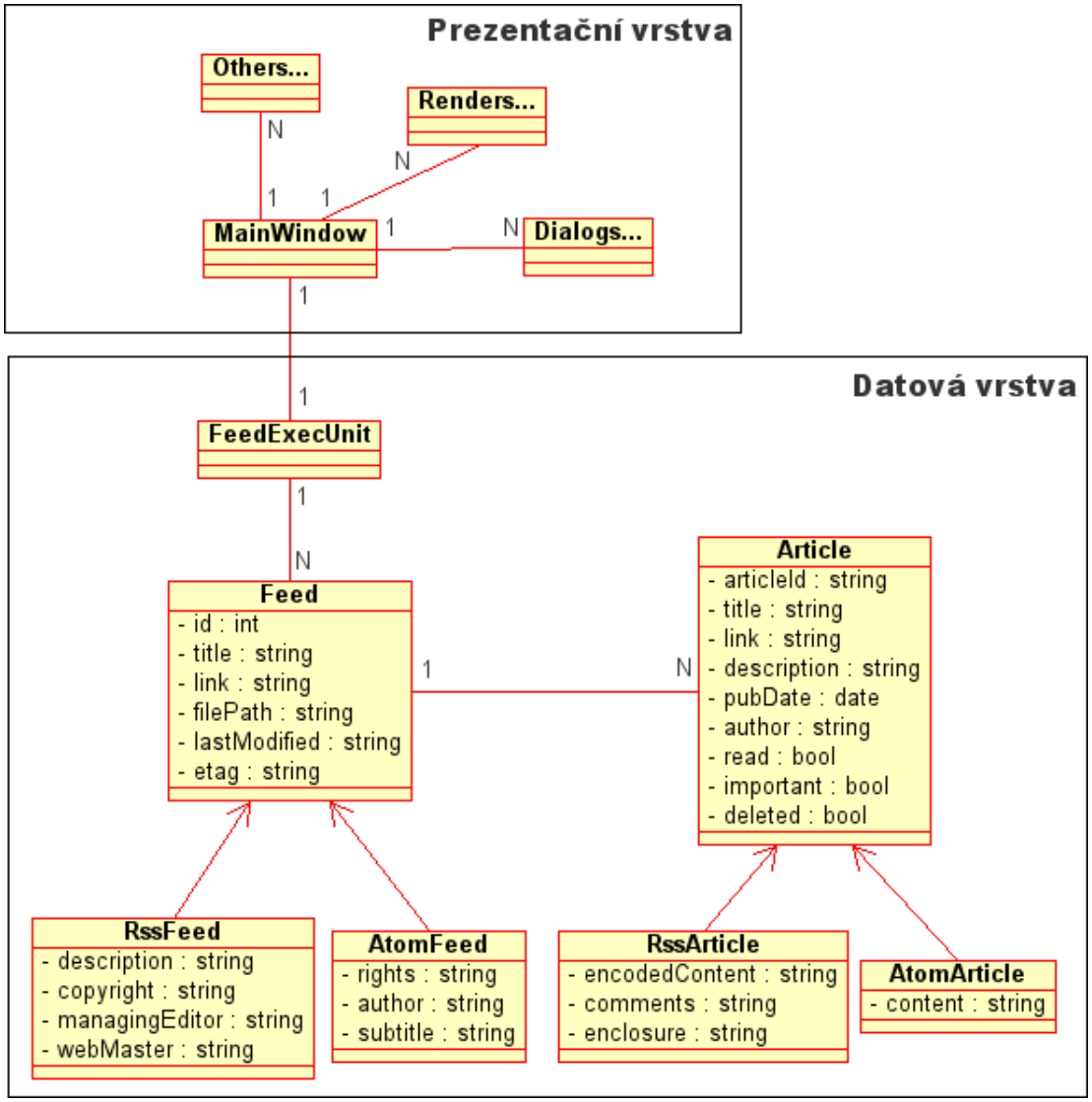
4.2.2 Datová vrstva

Do této skupiny patří všechny třídy, jejichž instance budou v paměti tvořit načtené kanály a jejich články.

Jeden článek je zde představován třídou `Article`. Obsahuje atributy pro uložení ID článku, názvu článku nebo odkazu na obsah na webu a jiné. Všechny tyto vlastnosti jsou společné pro oba typy zdrojů. Jak již víme z podkapitoly zabývající se strukturou kanálů, existují zde určitě rozdíly v metadatech, které nabízí články RSS a Atom. Z tohoto důvodu je vhodné rozšířit `Article` o další specializovanou třídu symbolizující konkrétní typ článku. Takto nám vznikly další třídy `RssArticle` a `AtomArticle`, které dědí všechny vlastnosti jejich otcovské třídy a dále ji specifikují. Kromě toho musí `Article` uchovávat ještě několik dalších atributů, které již nejsou spojeny s informacemi získanými z internetu, ale určují stav článku v dané programové relaci. Jak již bylo zmíněno dříve, každý článek může nabývat tří stavů. Z uživatelského pohledu se dá považovat za přečtený, nepřečtený, nebo důležitý. Tohle rozlišení zajišťují logické atributy `read` a `important`. Daný článek může být také smazán čtenářem s tím, že už dále nechce, aby se mu objevoval v seznamu. Ten sice zmizí ze zobrazené tabulky, avšak fyzicky smazán nebude. Pouze se mu nastaví vlastnost `deleted` na logickou hodnotu *true* a bude smazán jakmile vyprší jeho platnost (viz následující podkapitola). Kdyby byl totiž smazán okamžitě z datové struktury, hrozilo by zde riziko, že při další aktualizaci kanálu se opětovně stáhne a zobrazí v tabulce, což je nežádoucí.

Každý článek patří k určitému kanálu. Ten je v tomto projektu symbolizován třídou `Feed` a v prvé řadě obsahuje seznam vztahujících se k němu článků. Kromě jiného uchovává metadata s ním spojená, jako například ID, název kanálu, atd. Stejně tak, jak tomu bylo u článků, i v tomto případě je vhodné tuto obecnou třídu dále rozšířit o speciální vlastnosti nabízené RSS a Atomem. Díky tomuto návrhu jsou zde nechány otevřené dveře i pro případné další rozšíření aplikace o jiné typy kanálů, které mohou v budoucnu vzniknout a stát se populárními. Další atribut se nazývá `filePath` a uchovává relativní cestu k souboru s uloženým kanálem na disku. Zbylé dvě položky `lastModified` a `etag` je nutné uchovávat kvůli použití podmíněných GET požadavků při aktualizaci zdroje (viz kapitola Stahování kanálů).

Poslední nezmíněnou v této vrstvě je třída `FeedExecUnit`, která obsahuje seznam všech zdrojů a zastřešuje složitější operace nad kanály a články tak, aby se s nimi dalo pohodlně



Obrázek 4.2: Výsledný diagram tříd.

pracovat v hlavní třídě `MainWindow`. Její vlastnosti z důvodu úspory místa v obrázku nejsou zobrazeny, avšak nejsou ani příliš podstatné, abychom se jimi zde zabývali.

4.3 Archivace kanálů a uživatelských nastavení

Nyní, když už je hotov objektový model aplikace, zbývá ještě zajistit perzistenci určitých dat tak, aby se uživateli při opětovném spuštění načetla předešlá programová relace. V opačném případě by uživatel musel při každém spuštění aplikace znovu navolit všechny kanály a nastavení, což nepřichází v úvahu. Aplikace tedy musí vytvořit určitou databázi, ke které bude při svém spuštění a při uzavření přistupovat a provádět v ní příslušné operace. Pro tyto účely nám bude zcela stačit XML databáze, která je přehledná a vyskytuje se v čitelné podobě, takže se dá velmi jednoduše upravovat.

Jakmile uživatel dá příkaz ke smazání kanálu, ten bude z databáze odstraněn. Jakým

způsobem se ale budou mazat články? Musí zde existovat funkce, která bude provádět automatické odstranění starých zpráv, aby tento úkon nemusel dělat uživatel ručně. Vzhledem k tomu, že víme, kdy byl daný kanál aktualizován, můžeme nabídnout odebírateli možnost samostatného odstraňování článků, které jsou starší, než je nastaveno (vyprší jejich platnost). Ve většině aplikací se tomu říká operace údržby, která se provádí jednou za čas. V takovém případě je ale nutné ukládat na disk i tohle nastavení.

Kromě zmíněných kanálů, článků a nastavení archivace musí být uchovávány také uživatelské kategorie. Nejjednodušší způsob spočívá ve vytvoření XML souboru, do kterého se budou ukládat hierarchicky uspořádané kategorie a reference kanálů, které do nich patří. Při spuštění aplikace pak stačí načíst zmíněný soubor a podle referencí spojit kategorii s příslušným objektem.

Kapitola 5

Realizace projektu

Předcházející kapitola byla věnována návrhu aplikace a nyní se logicky dostáváme do další fáze vývoje, a to přímo k samotné implementaci. Následující text popisuje většinu problémů a jejich průběžné řešení, které se vyskytly v souvislosti s realizací inteligentní RSS čtečky.

5.1 Volba implementačního jazyka a vývojového prostředí

Návrh programu je dokončen a nyní je nutné určit, v jakém programovacím jazyce bude aplikace vyvíjena. Byl zde nastíněn určitý objektový model, takže je vhodné si zvolit objektově orientovaný jazyk, pomocí kterého by se dal tento návrh realizovat.

Jako jeden z hlavních požadavků na vlastnosti výsledné aplikace jsem si stanovil platformní nezávislost. Chtěl jsem, aby RSS čtečka mohla být spustitelná a dále rozšiřitelná na jakékoliv architektuře bez omezení. Pro implementaci aplikace jsem se rozhodoval mezi C++ nebo Javou. Oba zmíněné jazyky nabízí své speciální vlastnosti a širokou podporu tvorby GUI aplikací pomocí nejrůznějších toolkitů. Nakonec jsem zvolil jazyk Java, který nabízí mnohé užitečné nástroje, jako například systém vyvolávání vyjímek a jejich odchytávání, automatickou správu paměti v podobě *Garbage Collectoru* a v neposlední řadě také knihovny AWT a Swing pro tvorbu GUI poskytující poměrně pestrou paletu komponent a ovládacích prvků.

Nyní je třeba ještě zvolit prostředí, ve kterém bude aplikace vyvíjena. Nabízí se dvě nejznámější, tedy prostředí Netbeans od firmy Sun Microsystems (nyní už spíše Oracle) a prostředí Eclipse od Eclipse Foundation. Vzhledem k tomu, že jsem již z minulosti byl zvyklý pracovat s Netbeans IDE, dlouho jsem se nerozhodoval. Zmíněné prostředí nabízí vývojáři mnoho velmi propracovaných funkcí a možností. Hlavní výhodou oproti jiným IDE je automatická kontrola syntaxe a částečně i sémantiky přímo v průběhu psaní (díky JVM) a doplňování rozepsaného kódu. Existují i další funkce jako generování *JavaDoc* dokumentace, inteligentní vícevláknový debugger, profilování aplikace (získávání statistik o běhu jednotlivých vláken, zátěže procesoru a využití paměti) a jiné.

5.2 Stahování kanálů

Vzhledem k tomu, že chceme pracovat s daty poskytovanými různými servery v rámci internetu, je potřeba je nejprve nějakým způsobem získat. Stejně tak, jak je tomu například u webových stránek, soubor s RSS zprávami můžeme jednoduše od serveru stáhnout pomocí GET požadavku. Ten nám ve většině případů vyhoví a zašle zpět požadovaný XML

soubor a návratový kód 200 v HTTP hlavičce značící, že vše proběhlo v pořádku. Někdy se ovšem může stát, že nám z nějakého důvodu nevyhoví. Například pokud se požadovaný soubor na serveru již nenachází. V takovém případě nezbývá, než o tom informovat uživatele příslušným dialogem.

5.2.1 Podmíněný HTTP GET

Kapacity serverových stanic jsou dnes omezené a s postupem času stále přibývá požadavků, které musí zvládnout obsloužit. Také serverová síť bývá často přetížena. Z tohoto důvodu by aplikace na klientské straně měly svůj počet žádostí omezovat a nebo užívat takové techniky spojení, aby servery zbytečně nezatěžovaly.

Jedním z těchto používaných způsobů je právě podmíněný GET požadavek (anglicky *conditional GET request*). Ten funguje tím způsobem, že klientský program si při prvním stažení dokumentu někam uloží hodnoty `lastModified` a `etag`, které jsou mu předány v HTTP hlavičce od serveru. `lastModified` obsahuje datum poslední změny a `etag` uchovává unikátní hash dokumentu. Při příští aktualizaci se pak do HTTP hlavičky přidají dva parametry. Prvním je `If-Modified-Since`, do něj vložíme získanou hodnotu z `lastModified` a druhým `If-None-Match`, do kterého patří hash řetězec z `etag`. Server obdrží tento požadavek a porovná obě hodnoty se svými. Pokud jsou různé, zašle zpět normálním způsobem nejnovější verzi dokumentu. Pokud ovšem jsou stejné, odpoví klientovi v podobě návratového kódu 304 znamenající „žádná změna“ (anglicky *not modified*) a tím mu dává na vědomí, že má nejnovější verzi již k dispozici a ukončí spojení. Je důležité, aby oba tyto řetězce přesně odpovídaly tomu, co bylo získáno při prvním stažení, protože server, který bude následně zpracovávat tento požadavek, může používat různé metody pro porovnání těchto hodnot [12].

Protože i RSS čtečky jsou jedním z činitelů, kteří enormě a mnohdy zbytečně bombardují servery požadavky a plýtvají tak šířkou přenosového pásma, je na místě tento postup využívat. Lidé mívají často nastavané automatické aktualizace zdrojů v intervalech v řádech jednotek minut, což je pro server, který aktualizuje své novinky jednou denně, zbytečná zátěž. Z těchto důvodů je použit podmíněný GET požadavek při každé aktualizaci.

5.2.2 Inteligentní vyhledávání zdrojů

Uvažujme situaci, že uživatel si chce přidat nový zdroj k odebírání. V takovém případě klikne na příslušné tlačítko, nebo vybere akci z kontextového menu a v zobrazeném dialogu zadá URL adresu odkazující na konkrétní XML soubor na webu. Jenže co dělat v situaci, kdy uživatel nezná tuto adresu? Nebylo by asi správným řešením nechat jej zoufale prohledávat všechny odkazy na stránce. Čtečka by měla být schopna sama nalézt RSS zdroje na webu.

Webové stránky mívají v hlavičce seznam odkazů na soubory obsahující kaskádové styly, skripty, ale také RSS zprávy. Ten může vypadat následovně:

```
<link rel=”alternate” type=”application/rss+xml” title=”openSUSE”
    news &amp; events” href=”http://news.opensuse.org/feed/” />
```

Pomocí pár regulárních výrazů lze elegantně tuto adresu získat.

1. `<link[^<]*type=\”application/rss[^<]*>`
2. `href=\”([^\”]*)\”`

Prvním výrazem vytáhneme z HTML kódu řádek, který jako typ svého obsahu udává RSS. Tento řádek se dále zpracovává a pomocí druhého regulárního výrazu získáme URL odkaz na soubor.

Nyní máme k dispozici adresu, na které by se měl podle všeho nacházet zdroj zpráv. To ale nemusí platit vždy. Některé odkazy můžou směřovat na jiné HTML stránky s požadovanou URL. Jak tedy zjistit, zda jsme stáhli další HTML nebo už přímo XML soubor s novinkami? Odpověď se nachází v HTTP hlavičce, do které zapisuje server typ dokumentu pod parametr `Content-Type`. Pokud se jedná o HTML, rekurzivně opakujeme další stahování dokud nenarazíme na požadovaný RSS soubor. Jestliže nebyl nalezen, je třeba oznámit uživateli, že na zadané URL nebyl nalezen zdroj novinek.

5.2.3 Vícevláknová politika

Stahování dokumentů z internetu patří vůbec k nejpomalejším operacím aplikace závisejícím i na jiných technických okolnostech, které nemůže programátor ovlivnit. Připojení k internetu a jeho rychlost se na různých stanicích a místech liší, je tedy nutné, aby s tím aplikace počítala. Z tohoto důvodu musí program používat více vláken k vykonávání posloupnosti instrukcí. V opačném případě by docházelo k nepříjemnému „zamrznutí“ a čekání, než se příslušný kanál stáhne. Pomocí paralelního přístupu zpracování instrukcí může aplikace současně stahovat dokument a zároveň reagovat na další uživatelské akce.

Původní koncept byl takový, že pro každý momentálně stahovaný kanál se vytvoří nové vlákno a docílí se tak maximální rychlosti a efektivity. Avšak můžou nastat i situace, kdy uživatel chce stáhnout naráz opravdu velké množství zdrojů, přičemž by se vytvořilo tolik nových vláken, které by v důsledku toho značně zatížily procesor a jejich zpracování by se stalo pomalé a neefektivní. Je tedy nutné určitým způsobem limitovat počet běžících vláken. Toho lze docílit tak, že vlákno, které cyklicky vytváří ostatní vlákna určená ke stahování, na čas uspíme a po jisté době jej opět probudíme k vytvoření dalších vláken. Zde je potřeba nějakým způsobem uchovávat, kolik vláken je momentálně spuštěno. Pokud jich stále běží maximální určený počet, hlavní vlákno nepokračuje v tvorbě dalších. Zmíněný počet lze mít v podobě synchronizovaného atributu, jehož hodnotu každé nové vlákno zvýší o jedna a před ukončením své činnosti jej dekrementuje.

V Javě existují metody zvané `wait()` a `notify()`, které zajišťují funkce pro uspání a probuzení vlákna. Metoda `wait()` může obsahovat volitelný parametr udávající dobu v milisekundách, po kterou má být vlákno uspáno. Lze ji ovšem použít pouze v tzv. synchronizovaném bloku. Jedná se o část kódu, do které může „vstoupit“ ve stejný čas pouze jedno vlákno. V podstatě jde o obdobu programových zámků. Takto uspané vlákno může být probuzeno pomocí jiného vlákna, které zavolá `notify()` a nebo po vypršení zadaného časového intervalu [11].

5.3 Parsování kanálů

Nyní, když už dokážeme stáhnout XML soubor s požadovaným kanálem, ještě zbývá získat z něj příslušná data a metainformace. Takové operaci se říká parsování (z anglického *parse* - analyzovat, rozebrat). Můžeme opět využít regulárních výrazů, avšak existují i pohodlnější způsoby. Na internetu lze najít velké množství externích knihoven, které byly vytvořeny speciálně pro parsování a operace s XML. Všechny tyto vychází ze dvou základních přístupů ke zpracování XML:

- **SAX** - neboli *Simple API for XML*, je velmi dobře postaveným rozhraním pro zpracování XML. Tento parser poskytuje i mnohé pokročilé funkce a možnosti. SAX je událostmi řízené API. Tyto události se chovají v podstatě stejně jako u AWT a Swingu. Díky tomu je SAX velmi rychlý a paměťově nenáročný. Na druhou stranu ovšem neposkytuje žádnou třídu, která by získaná data reprezentovala jako celek. Je tedy nutné, aby si progamátor vytvořil své vlastní datové struktury.

- **DOM** - je zkratkou pro *Document Object Model* a je nejznámějším prostředkem pro operace s XML. Není vázán na konkrétní programovací jazyk a lze jej použít téměř kdekoliv. Celé XML modeluje pomocí stromové struktury, přičemž každý soubor je jeden `Document object`. Nabízí další metody pro práci s XML. Narozdíl od SAX, podporuje jak čtení, tak zápis. Nicméně jeho použití je poměrně paměťově náročné v porovnání se SAX.

Mezi nejznámější knihovny pro parsování XML v Javě pak patří:

- **JAXP** - znamená *Java API for XML Processing*. Tohle rozhraní nabízí oba zmíněné přístupy svázané dohromady. Od verze Java 1.4 je její standardní součástí. Před prvním použitím je třeba se rozhodnout, kterou technologii uplatnit, zda SAX, nebo DOM.

- **JDOM** - je pseudoakronym pro *Java Document Object Model*. JDOM je rozhraní navržené speciálně pro jazyk Java, které není parserem, ale Java reprezentací XML dokumentu. Může používat v podstatě jakýkoliv parser. Jako výchozí je nastaven ten, který používá JAXP. JDOM dokument může být vytvořen z XML souboru, DOM stromu nebo ze SAX událostí. Snaží se odstranit mnohé nedostatky DOMu a maximálně zjednodušit práci vývojářům. Používá standardní Java třídy, konstruktory a metody. Stejně tak jako DOM, nejprve načte celý dokument do paměti a pak jej zpracovává.

- **DOM4J** - původně součástí JDOM projektu, avšak později se odtrhnul. Stejně tak jako JDOM, je určen pouze pro jazyk Java, avšak místo konkrétních tříd používá rozhraní. Obsahuje některé speciální vlastnosti jako podporu pro XPATH a XSLT.

- **ElectricXML** - je velmi jednoduché a snadno ovladatelné API. Avšak nepracuje zcela korektně co se týče jmenných prostorů. Další podstatnou nevýhodou je, že nepatří do kategorie open source, a tudíž je jeho použitelnost omezená.

- **XMLPULL** - patří do skupiny tzv. pull parserů. Ty se od ostatních liší tím, že zpracovávají danou část XML pouze, až si to aplikace vyžádá (vhodné pro velmi rozsáhlé XML dokumenty). Nicméně má i několik nedostatků. Jedním z nich je to, že podpora jmenných prostorů je ve výchozím nastavení vyplá nebo také ignorování `DOCTYPE` deklarace [8].

Po zvážení všech aspektů jsem se nakonec rozhodl pro knihovnu JDOM, která je jednoduchá na používání a poskytuje mnoho metod nejen pro čtení z XML, ale i pro zápis. Po vyparsování prvků z XML jsou tyto informace dostupné v podobě hierarchického stromu. Jednotlivé uzly stromu jsou zde nazvány elementy, které můžou obsahovat další uzly, atributy nebo jmenné prostory. Pomocí metod `getChild()` nebo `getChildren()` se můžeme dostat k příslušným poduzlům. Metodou `getChildText()` pak získáme hodnotu konkrétní

značky, jejíž jméno předáme v parametru. Pokud chceme vytáhnout obsah značky z jiného než výchozího jmenného prostoru, musíme v druhém volitelném parametru předat instanci třídy `Namespace`, která jej reprezentuje. Ta může být vytvořena voláním metody `getNamespace()` nad konkrétním uzlem, kde je příslušný jmenný prostor deklarován.

Jak se dozvíme později, aplikace využívá i externího pull parseru kvůli rychlé serializaci a deserializaci dat z disku.

5.3.1 Převedení datumu a času na místní nastavení

Ze struktury kanálů pro syndikaci obsahu již víme, že pod značkou `<pubDate>` nebo `<published>` se nachází časová známka udávající, kdy byl kanál zveřejněn (u článků Atomu většinou `<published>` chybí, avšak je zde element `<updated>` zohledňující poslední aktualizaci, který můžeme zpracovávat stejným způsobem). Hodnoty těchto prvků bývají uvedeny ve standardizovaných formátech. RSS používá RFC 822, jehož fyzická podoba může vypadat následovně:

- Sun, 02 May 2010 19:36:59 -0400
- Sun, 02 May 2010 23:36:59 GMT
- Sun, 02 May 2010 19:36:59 EDT

Abychom mohli správně převést datum na naše časové poměry, musíme si vysvětlit značky, které se vyskytují v RSS a identifikují časové pásmo, ve kterém byl kanál napsán.

- GMT - *Greenwich Mean Time* je považován za výchozí bod a ostatní časová pásma se od něj dají určit přičtením nebo odečtením konkrétním počtem hodin.
- UTC - *Universal Time Coordinate* je pouze jiným označením pro GMT.
- Pro zimní čas platí:
  - EST - *Eastern Standard Time* (UTC−5 hodin)
  - CST - *Central Standard Time* (UTC−6 hodin)
  - MST - *Mountain Standard Time* (UTC−7 hodin)
  - PST - *Pacific Standard Time* (UTC−8 hodin)
- Pro letní čas platí:
  - EDT - *Eastern Daylight Time* (UTC−4 hodiny)
  - CDT - *Central Daylight Time* (UTC−5 hodin)
  - MDT - *Mountain Daylight Time* (UTC−6 hodin)
  - PDT - *Pacific Daylight Time* (UTC−7 hodin)

Atom na druhou stranu používa standard RFC 3339:

- 2010-03-02T23:36:59Z

Velké písmeno T zde figuruje pouze jak oddělovač datumu a času. Velké Z pak znamená „Zulu“ používané v armádě jako označení centrálního časového pásma (znamená to samé co GMT).

Pokud chceme zobrazovat správný datum zveřejnění u každého článku, musíme získanou hodnotu převést na naše časové pásmo. Pro Českou republiku platí v zimním období čas UTC+1 hodina a v letním UTC+2 hodiny. Jazyk Java nabízí poměrně rozsáhlou strukturu tříd operujících s kalendáři a datumy používanými v nejrůznějších částech světa. Mezi užitečné patří i třída `SimpleDateFormat`, která v sobě skrývá metody pro parsování formátů datumu. Nejprve je třeba sestavit řetězec se vzorkem, podle kterého se bude datum parsovat. Poněvadž `SimpleDateFormat` nezná označení Zulu, musíme všechna velká Z nahradit zkratkou GMT. Nyní je třeba sestavit parsovací vzorek, ve kterém má každe písmeno svůj speciální význam. Více informací o tvorbě vzorků lze nalézt v online referenci [2]. Protože v RSS může být datum zapsán více různými formáty, musíme pomocí kaskády bloků `try, catch` zkoušet načíst datum přes více vzorků.

- “`EEE, d MMM yyyy HH:mm:ss Z`” pro formát: Sun, 02 May 2010 19:36:59 -0400
- “`EEE, d MMM yyyy HH:mm:ss z`” pro formát: Sun, 02 May 2010 23:36:59 GMT
- “`yyyy-MM-dd'T'HH:mm:ssz`” pro formát: 2010-03-02T23:36:59GMT

Tímto způsobem jsme získali datum v podobě objektu třídy `Date`, který je již převeden do našeho časového pásma.

5.4 Základní komponenty GUI

Hlavní menu bylo podle návrhu rozděleno do základních tří sekcí, přičemž každá z nich obsahuje více jednoduchých či složitých komponent. Za složité komponenty považujme takové, které potřebují pro svou správnou funkci další externí třídy. Například pro renderování visuální podoby, operace s datovým modelem a jiné. Nyní se podívejme na některé z nich.

5.4.1 Strom uživatelských kategorií

Je tvořen komponentou *JTree*, která je určena právě pro případy, kdy chceme ve své aplikaci použít stromový pohled na určitá data. Tento stěžejní ovládácí prvek poskytuje abstrakci pro uživatele ve formě rozdělení kanálů do příslušných kategorií. Ve skutečnosti jsou kanály stále ve stejné kolekci a z *JTree* se na ně pouze pomocí referencí odkazujeme. Díky tomu je zajištěna jednoduchá manipulace se stromem. Pokud je daný uzel přemístěn na jinou pozici, nepřesunují se fyzicky příslušné kanály, ale pouze objekty odkazující se na ně.

Byla vytvořena třída `ReaderTreeCellRenderer`, která zajišťuje vykreslování jednotlivých uzlů stromu. V podstatě dědí všechny vlastnosti od výchozí třídy `DefaultTreeCellRenderer`, avšak rozšiřuje její metodu pro renderování uzlů tím způsobem, že ke každému listovému uzlu vykresluje ikonu s RSS logem.

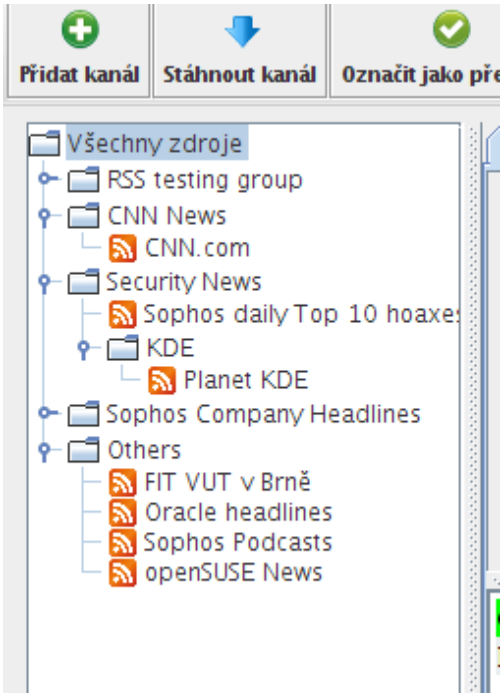
Poměrně rozsáhlá a důležitá je třída `TreeViewParser`, která má na starosti veškeré operace s datovým modelem stromové komponenty. Úzce spolupracuje se zmíněnou třídou `FeedExecUnit` a poskytuje metody pro načtení nebo uložení uživatelských kategorií pomocí JDOM do XML, přidání nového kanálu, přidání složky, stažení označeného zdroje a jiné.

Nyní je již komponenta jak po datové, tak po visuální stránce připravena. Avšak ještě schází připojit podporu pro funkce *Drag & Drop*, tedy přetahování uzlů pomocí myši. Tuto

úlohu vykonává třída `TreeTransferHandler`, která rozšiřuje původní třídu `TransferHandler` a překrývá několik jejich stěžejních metod. V konstruktoru zmíněné třídy je vytvořen řetězec pro určení, které MIME objekty je možno přenášet. Mezi nejdůležitější metody patří například: `getSourceAction()` určující, jaký typ akce je povolen. K těm patří přesouvání, kopírování, nebo obojí (v našem případě pouze přesouvání). Překrytá metoda `ExportDone()` udává, jaké operace se mají provést po úspěšném exportu DND (v tomto případě vymazání uzlu z původního místa stromu). Následuje metoda `canImport()`, která je volána pokaždé, kdy se uživatel přetahující daný uzel dostane myší nad jiný uzel. Vrací *true*, nebo *false* v závislosti na tom, zda uživatel smí takovou operaci provést. V těle metody `canImport()` jsou naprogramovány pravidla, která určují návratovou hodnotu. V tomto kontextu byla vytvořena dvě pravidla:

1. Přesunutí uzlu na stejné místo je zakázáno.
2. Přesunutí uzlu, který není listem, na nižší úroveň svého vlastního podstromu je zakázáno.

V závislosti na zmíněných pravidlech se uživateli vedle kurzoru objeví odpovídající ikona. Poslední důležitá překrytá metoda nese název `importData()` a je určena pro načtení importovaných dat (zařazení uzlu na příslušné místo). Na obrázku 5.3 pak můžeme vidět konečnou podobu této komponenty.



Obrázek 5.1: Komponenta s přehledem kanálů a kategorií

5.4.2 Tabulka s články

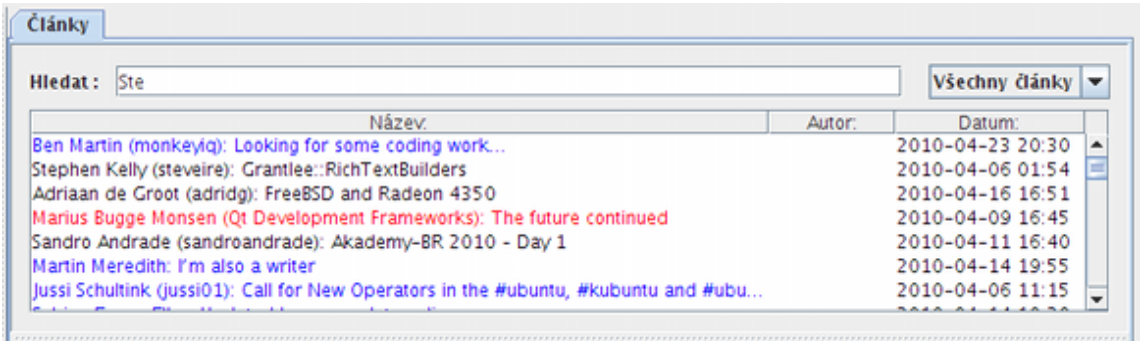
Má za úkol zobrazovat články splňující filtrovací a vyhledávací kritéria. Fyzicky je reprezentována komponentou *JTable*. Jak již bylo zmíněno dříve, musí nějakým způsobem visuálně

odlišit články podle jejich skupin. K tomu ji dopomáhá třída `ArticleTableRenderer` a její překrytá metoda pro vykreslení buňky tabulky. Přečtené články jsou označeny černou barvou písma, nepřečtené modrou a důležité červenou barvou.

Tabulka obsahuje celkem tři sloupce. V prvním je pochopitelně zobrazen název článku, následuje jeho autor (pokud je uveden) a datum. Články lze podle očekávání třídit vzestupným i sestupným způsobem v závislosti na vybraném sloupci.

Každé pole ovládacího prvku *JTable* může nést uživatelský objekt jakéhokoliv typu. Stejně tak, jak tomu bylo u *JTree* i zde je tímto objektem pouze odkaz na příslušný článek. Jak již bylo zmíněno v předešlé kapitole, pokud je provedena akce odstranění článku, objektu, který jej reprezentuje, se nastaví vlastnost `isDeleted` na hodnotu *true* a při následném překreslení tabulky je vyloučen ze zpracování.

K této komponentě neodmyslitelně patří také dva jednodušší prvky poskytující funkce spojené s vyhledáváním. Jedná se o textové pole pro zadání hledaného výrazu, které je realizováno pomocí *JTextField* a roletové menu pro výběr skupiny, do kterého má hledaná zpráva patřit. Na zmíněné textové pole byl nasazen *listener* (obsluha události) zachytávající každou událost spojenou se stiskem klávesy uvnitř pole. Pomocí něj je tabulka během psaní okamžitě překreslována vyhovujícími články.



Obrázek 5.2: Komponenta pro zobrazení tabulky s články.

5.4.3 Panel s popisem či obsahem článku

V závislosti na označeném článku v tabulce se v pravém spodním okraji zobrazí jeho obsah. Tímto je u RSS myšlen text prvku `<encoded:content>`, ve kterém většinou bývá uchován kompletní obsah zprávy i s HTML formátováním. Pokud zmíněný element chybí, vypíše se popis článku z prvku `<description>`. U Atomu je tomu nápodobně. Pokud je obsažena značka `<content>`, zobrazí se zde její obsah, pokud není, vypíše se text prvku `<summary>`.

Jak již bylo zde vícekrát řečeno, zmíněné značky většinou obsahují HTML formátování. Pro zobrazení takového obsahu byla použita komponenta *JEditorPane*, která jej dokáže vhodně vykreslit. K původnímu HTML kódu je vždy na začátek doplněna jednoduchá tabulka fungující jako záhlaví článku s jeho názvem a datumem. Na konec zprávy je připojeno několik hypertextových odkazů směřujících čtenáře na kompletní obsah článku na webu, komentáře k článku a v neposlední řadě je zde také odkaz na případné MIME přílohy. Všechny hypertextové odkazy jsou funkční a při jejich aktivaci je volán výchozí externí prohlížeč s odpovídající URL.

Obrázek 5.3: Komponenta pro zobrazení popisu článku.

5.4.4 Zobrazení webové stránky v aplikaci

Dalším neméně ambiciózním cílem během vývoje bylo zobrazení kompletní webové stránky přímo v programu. Za takových okolností by uživatel nemusel být z hlavního menu hypertextově odkazován na externí prohlížeč, ale mohl by si konkrétní stránky zobrazit přímo v čtečce.

K tomuto účelu byl použit jeden z projektů zabývající se renderováním HTML stránek pod názvem *Cobra*. Jedná se o čistě Javovský engine k zobrazování webových stránek, který má podporu pro HTML 4, Javascript a CSS 2. Tento engine je součástí většího projektu zvaného *Lobo*, který je již poměrně schopným webovým prohlížečem podporujícím cacheování, cookies, HTTP a HTTPS autentifikaci a jiné techniky. Zmíněný projekt je poměrně rozsáhlý a pro účely této aplikace zbytečně robustní. Vzhledem k tomu, že v našem případě je potřeba pouze zobrazovat webové stránky, je *Cobra* zcela postačujícím prostředkem [5]. Avšak je třeba zmínit, že se jedná stále o projekt ve fázi vývoje, který nemůže plně suplovat zobrazení profesionálním internetovým prohlížečem a nedokáže zobrazit zcela korektně všechny efektově náročně stránky.

Původní záměr byl takový, že při klepnutí na hypertextový odkaz v panelu s popisem článku se otevře příslušná stránka v nové záložce. Avšak vzhledem k náročnosti zobrazení některých stránek byla výchozí volba nastavena na externí prohlížeč. Stránku lze i přesto zobrazit pomocí *Cobra* enginu v nové záložce přes tlačítko „Zobrazit obsah“ nebo příslušnou volbou z kontextového menu článku.

5.5 Uchování poslední relace

Z návrhu vyplývají požadavky na persistenci některých objektů, jejichž životnost musí zůstat zachována i po uzavření aplikace tak, aby při opětovném spuštění byla načtena původní relace. Celkově je nutné ukládat následující data:

1. Kanály a jejich články, které uživatel odebírá.

2. Konkrétní nastavení archivace článků.

3. Strukturu uživatelských kategorií obsahující odkazy na příslušné zdroje.

Jako první možnost pro uchovávání kanálů na disku se jeví použití zápisových metod rozhraní JDOM. V Javě však existuje i jednodušší způsob pro uložení konkrétních objektů na disk. Nazývá se serializace. Pod tímto termínem je myšleno převedení určitých datových struktur do podoby, která je uzpůsobena k zápisu na disk a kterou je možno znovu načíst zpět do paměti. Tvůrci jakyka Java k tomuto účelu vytvořili rozhraní `Serializable`, pomocí kterého lze třídu, která jej implementuje, převést do binárního tvaru a uložit na disk. Velmi podobným způsobem lze daný objekt uložit do podoby XML. Tento proces může při velkém množství ukládaných dat zabrat určitý čas. Nezapomínejme, že operace s pevným diskem jsou vždy relativně časově náročné. Na internetu lze nalézt knihovny přímo specializované na serializaci objektů do XML a poskytující velmi rychlé zpracování i u velkého množství ukládaných dat.

K tomuto účelu byla použita knihovna *XStream* nabízející velmi slušnou bilanci co se týče rychlosti převedení do XML a nízké paměťové náročnosti [6]. Její použití se dá považovat za velmi jednoduché, avšak pro zrychlení sestavování XML je doporučeno použít parser XPP3 patřící do kategorie pull parserů, o kterých již byla řeč. Nejprve je třeba vytvořit instanci třídy `XStream`, jejíž bezparametrový konstruktor předpokládá použití zmíněného parseru. Lze použít i jiný parser, například DOM, který se uvede v parametru. Následně lze metodou `toXML()` provést převedení do XML, v jejímž parametru musí být uveden serializovaný objekt a pomocí třídy `PrintWriter` zapsat tento výstup do souboru. Narozdíl od implicitních Javovských knihoven, *XStream* ukládá jak veřejné, tak soukromé atributy objektů. Každý kanál v kolekci je kvůli přehlednosti a snadné manipulaci uložen do samostatného souboru. Opětovné načtení se provádí analogickým způsobem pomocí metody `fromXML()`.

Další součástí programu, kterou je nutno uložit na disk, je nastavení archivace článků. Třída reprezentující tato nastavení se jmenuje `ArchiveSettings` a obsahuje jen několik málo atributů udávajících způsob a časový interval pro archivaci zpráv. Stejně jako v předešlém případě využijeme rozhraní *XStream* a serializujeme objekt zmíněné třídy do souboru `settings.xml`.

Poslední věcí, kterou je třeba při spuštění aplikace znovu načíst z disku jsou uživatelské kategorie. Knihovna *XStream* je schopna serializovat i celou GUI komponentu, což by se dalo použít i v tomto případě, avšak výsledný soubor by byl zbytečně velký a složitě strukturovaný na to, co z něj potřebujeme znát. Raději použijeme rozhraní JDOM a jeho metody pro vytvoření XML dokumentu. Základním elementem je zde prvek `<feed>` symbolizující referenci na konkrétní kanál. Vždy zahrnuje dva atributy. Prvním je `id` identifikující kanál a druhým `path` uchovávající relativní cestu k XML souboru se zdrojem. Značka `<feed>` je pak součástí prvku `<category>` reprezentující příslušnou kategorii, do které kanál spadá. Vše je ve výsledku uloženo do souboru `user.xml`.

5.6 Testování aplikace

Po úspěšné implementaci většiny funkcí se dostáváme do fáze testování, kdy jsou prověřovány všechny možnosti, za kterých by se aplikace mohla vymknout očekávanému chování. Nejprve bylo třeba ošetřit nedostupnost některých ovládacích prvků, například když není označen žádný článek, není možné stisknout tlačítka „Zobrazit obsah“ nebo „Zobrazit v

prohlížeči“. Stejně tak je tomu u akcí spouštěných z hlavního menu. Za takových okolností je celé menu „Článek“ znepřístupněno. Všechny tyto drobnosti by mohly vést k vyvolání neošetřené vyjímky a pádu programu. Uživatelské vstupy v dialozích jsou rovněž kontrolovány a v neposlední řadě bylo potřeba ošetřit možnosti práce s kořenovým uzlem komponenty *JTable*, který je nemodifikovatelný.

Byly zjištěny také některé chyby související s vyhledáváním zdrojů na zadané URL. Některé stránky používají ve svých odkazech relativní cesty, se kterými se původně nepočítalo. Zmíněný problém byl odstraněn doplněním celé adresy.

Vzhledem k požadavku na platformní nezávislost přichází na řadu testování na různých operačních systémech.

5.6.1 Operční systém Linux

Aplikace byla vyvíjena na operačním systému Linux verze `2.6.27.45-0.1`, distribuce openSuse. Podle předpokladu by měla pracovat korektně i na jiných distribucích. Avšak je třeba vše řádně vyzkoušet.

K tomuto účelu bylo využito školního systému Linux distribuce CentOS. Testován byl jak překlad pomocí nástroje *Ant*, tak spuštění výsledného jar souboru. Při prvním pokusu o sestavení jar souboru proběhlo vše v pořádku, avšak bylo třeba dokopírovat externí knihovny do složky lib. Při spuštění aplikace všechny funkce pracovaly správně. Nicméně při opětovném startu se nepodařilo načíst některé soubory s uloženým kanálem. Vzhledem k tomu, že tyto soubory byly pojemnovávány podle názvů kanálů, u českých zdrojů zde docházelo k problémům s kódováním. Tento problém byl jednoduše odstraněn generováním názvů souborů podle jejich *ID*.

5.6.2 Operační systém Windows

Vytvořený jar archiv byl testován také na operačním systému Windows XP. Program byl korektně spuštěn, avšak vyskytl se zde problém se zobrazováním kontextového menu. Při stisknutí pravého tlačítka myši nad příslušným článkem bylo provedeno jeho označení, avšak menu se neobjevilo. Tuto funkci mají na starosti potomci třídy `MouseAdapter`, která poskytuje několik metod k předefinování za účelem obsloužení událostí vyvolaných pomocí myši. Zmíněná událost také nastavuje hodnotu `popupTrigger` indikující stisk tlačítka pro zobrazení kontextového menu. U OS Linux je tento atribut nastaven hned při stisku tlačítka, což zajišťuje metoda `mousePressed()`, avšak u OS Windows je `popupTrigger` nastaven až při upuštění tlačítka, tudíž bylo nutné doimplementovat i metodu `mouseReleased()`, čímž byl problém vyřešen.

Kapitola 6

Závěr

Cílem práce bylo vytvoření aplikace pro stahování a agregaci internetových zpráv v podobě formátů určených k syndikaci obsahu. Program jako takový měl poskytovat uživateli komfortní ovládání pomocí grafického uživatelského prostředí s důrazem na intuitivnost, jednoduchou správu a přehlednost. Základním předpokladem bylo, že člověk pracující s touto aplikací je navyklý na určité ovládací standardy, které musí zůstat zachovány.

Nejprve bylo nutné důkladně nastudovat strukturu nejpoužívanějších syndikačních formátů, tedy RSS a Atom tak, abychom mohli využívat dat a metainformací, které poskytují.

Následně jsem vytvořil návrh uživatelského rozhraní v závislosti na zamýšlených funkcích a možnostech. Komponenty hlavního menu jsou logicky uspořádány tak, aby se uživatel nemusel učit novým návykům a aby program navenek působil přehledným a profesionálním dojmem. Byl navrhnut systém kategorií, podle kterého je možno členit kanály a články do menších a lépe spravovatelných skupin.

Po uvážení všech předcházejících poznatků byla vytvořena struktura jednotlivých tříd a jejich vazby. Za použití principů objektově orientovaného programování jsem sestavil výsledný diagram tříd, který fungoval jako základní konstrukce při následujícím vývoji aplikace.

V závislosti na takto vytvořeném návrhu jsem začal s realizací programu. Nejprve byl vybrán implementační jazyk a vývojové prostředí, v němž byla aplikace tvořena. Poté jsem se zabýval otázkou efektivního vyhledávání a získávání zdrojů zpráv z internetu, které je prováděno paralelně pomocí více vláken. Dalším problémem bylo parsování různých verzí těchto zdrojů a jejich interpretace uživateli v přehledné a strukturované formě. Funkce jednotlivých ovládacích prvků GUI jsem se snažil realizovat tak, aby byla práce s programem efektivní a jednoduchá, bez ohledu na kvantitu archivovaných článků.

Od počátku jsem chtěl vytvořit program, který nebude mít žádné bariéry v podobě různých architektur a operačních systémů. Z tohoto důvodu bylo provedeno testování i na jiných platformách, než na které byla aplikace vyvíjena a které po drobných úpravách dopadlo úspěšně.

6.1 Možnosti dalšího vývoje

Důraz byl kladen také na jednoduchou rozšiřitelnost, kterou jsem bral v úvahu už během návrhu. Aplikaci je možno upravit tak, aby byla schopna pracovat i s jinými typy zdrojů než RSS a Atom. Stačí implementovat metodu pro parsování takového kanálu a vytvořit třídu uchovávající jeho speciální vlastnosti, která by byla potomkem již připravené obecné

třídy reprezentující jeden zdroj.

Mezi další funkce, které by mohly čtečku zdokonalit, patří například import a export uživatelských nastavení do formátu OPML. Jedná se v podstatě o speciálně navržený XML dokument k výměně specifických dat mezi aplikacemi, které mohou běžet na různých operačních systémech a prostředích [16]. Pomocí takové operace by bylo možno jednoduše přenášet své oblíbené zdroje novinek na jiné aplikace stejného druhu.

Jako užitečné se jeví i klávesové zkratky, obzvláště při intenzivním používání programu. Jejich implementace a zařazení do menu nastavení by bylo také užitečným přínosem, přičemž by stačilo doplnit několik metod, které by obsluhovaly události při stisknutí konkrétní kombinace kláves.

Čtečka by mohla být rozšířena i o další funkce, mezi které například patří automatické stahování kanálů v daných časových intervalech, možnost interakce s webovými aplikacemi podobného zaměření nebo nastavení vzhledu uživatelského rozhraní z několika nabízených motivů (pomocí *Java Look & Feel*).

Literatura

[1] SOAP Meets RSS[online]. http://cyber.law.harvard.edu/rss/soapMeetsRss.html, 2008-04-02 [cit. 2010-05-03].

[2] Class SimpleDateFormat[online]. http://java.sun.com/javase/6/docs/api/java/text/SimpleDateFormat.html, 2009 [cit. 2010-04-27].

[3] Atom[online]. http://ec.europa.eu/ipg/standards/markup/web-content-syndication/atom/index_en.htm, 2010-04-29 [cit. 2010-05-03].

[4] History of RSS [online]. http://www.rss-specifications.com/history-rss.htm, 2010 [cit. 2010-04-13].

[5] The Lobo Project[online]. http://lobobrowser.org/index.jsp, [cit. 2010-05-03].

[6] XStream[online]. http://xstream.codehaus.org/index.html, [cit. 2010-05-03].

[7] BRADLEY, N.: *XML: kompletní průvodce*, ročník 1. Grada Publishing, spol. s.r.o., 2000, ISBN 80-7169-949-7, 540 s.

[8] HAROLD, E. R.: *Processing Xml With Java*. Addison Wesley, 2002-11-08, ISBN 0-201-77186-1, 1120 s.

[9] HOLZNER, S.; ŠINDELÁŘ, J.: *RSS: automatické doručování obsahu vašich WWW stránek*. Computer Press, a.s., první vydání, 2007, ISBN 978-80-251-1479-7.

[10] HRUŠKA, T.; BURGET, R.: Internetové aplikace (WAP) II., únor 2007, interní materiál FIT VUT v Brně.

[11] JENKOV, J.: Thread Signaling[online]. http://tutorials.jenkov.com/java-concurrency/thread-signaling.html, [cit. 2010-04-15].

[12] MILLER, C.: HTTP Conditional Get for RSS Hackers [online]. http://fishbowl.pastiche.org/2002/10/21/http_conditional_get_for_rss_hackers/, 2002-10-21 [cit. 2010-04-16].

[13] NOTTINGHAM, M.; SAYRE, R.: The Atom Syndication Format [online]. http://atompub.org/rfc4287.html, 2005-12-01 [cit. 2010-04-15].

[14] RUBY, S.; SNELL, J.: RSS 2.0 and Atom 1.0 Compared [online]. http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared#major, 2008-09-10 [cit. 2010-04-15].

[15] SOOD, V.: The world of Syndication: Atom 1.0 vs. RSS 2.0?[online]. http://blogs.iis.net/vsood/archive/2008/10/06/the-world-of-syndication-atom-1-0-vs-rss-2-0.aspx, 2008-10-06 [cit. 2010-05-02].

[16] WINER, D.: OPML[online]. http://www.opml.org/about, 2000-11-07 [cit. 2010-05-03].

Seznam použitých zkratek

RSS	*RDF Site Summary, Rich Site Summary, Really Simple Syndication*
RDF	*Resource Description Format*
HTML	*HyperText Markup Language*
XHTML	*eXtensible HyperText Markup Language*
XML	*eXtensible Markup Language*
W3C	*World Wide Web Consorcium*
DTD	*Document Type Definitions*
IETF	*Internet Engineering Task Force*
MIME	*Multipurpose Internet Mail Extensions*
IANA	*Internet Assigned Numbers Authority*
GUI	*Graphical User Interface*
DND	*Drag & Drop*
AWT	*Abstract Windowing Toolkit*
IDE	*Integrated Development Environment*
JVM	*Java Virtual Machine*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol - Secure*
URL	*Uniform Resource Locator*
URI	*Universal Resource Identifier*
API	*Application Programming Interface*
SAX	*Simple API for XML*
DOM	*Document Object Model*
JAXP	*Java API for XML Processing*
JDOM	*Java Document Object Model*
OPML	*Outline Processor Markup Language*
GMT	*Greenwich Mean Time*
UTC	*Universal Time Coordinate*
EST	*Eastern Standard Time*
CST	*Central Standard Time*
MST	*Mountain Standard Time*
PST	*Pacific Standard Time*
EDT	*Eastern Daylight Time*
CDT	*Central Daylight Time*
MDT	*Mounain Daylight Time*
PDT	*Pacific Daylight Time*

Seznam příloh

Dodatek A

Programová dokumentace

Kompletní programová dokumentace ve formě *JavaDoc* je k dispozici na přiloženém CD (viz dodatek B).

Dodatek B

CD se zdrojovými kódy a spustitelnou verzí aplikace

Obsah kořenové složky:

`/src`	zdrojvé kódy aplikace
`/dist/RssReader.jar`	sestavený program připravený ke spuštění
`/dist/javadoc/index.html`	programová dokumentace ve formě *JavaDoc*
`/dbdata`	vzorová databáze
`readme.txt`	pokyny k opětovnému sestavení programu
`/tex`	zdrojové soubory LaTeXk technické zprávě
`/tex/RssReader.pdf`	technická zpráva